

## A SUCCESSFUL ALGORITHM FOR SOLVING DIRECTED HAMILTONIAN PATH PROBLEMS

Gerald L. THOMPSON and Sharad SINGHAL

*Management Science Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA 15213, USA*

Received June 1983

Revised November 1983

In this paper we present a graph-theoretic polynomial algorithm which has positive probability of finding a Hamiltonian path in a given graph, if there is one; if the algorithm fails, it can be rerun with a randomly chosen starting solution, and there is again a positive probability it will find an answer. If there is no Hamiltonian path, the algorithm will always terminate with failure. We call this a *Successful Algorithm* because it has high (close to 1) empirical probability of success and it works in polynomial time. Some basic theoretical results concerning spanning arborescences of a graph are given. The concept of a ramification index is defined and it is shown that ramification index of a Hamiltonian path is zero. The algorithm starts with finding any spanning arborescence and by suitable pivots it endeavors to reduce the ramification index to zero. Probabilistic properties of the algorithm are discussed. Computational experience with graphs up to 30000 nodes is included.

Hamiltonian path problem • ramification index • polynomial algorithm

### 1. Introduction

The Hamiltonian cycle problem is the problem of finding a path in a graph which passes through each node exactly once. This problem is well known and has been discussed in most graph theory books such as [1,3,5].

A polynomial algorithm will be presented in this paper which has positive probability of finding a Hamiltonian path in a given graph if there is one. If a graph has a Hamiltonian path, this algorithm will either find it or end with a message that it cannot proceed further. However, if there is no Hamiltonian path in the graph, the algorithm

will always end with the 'cannot proceed further' message.

Despite the fact that the algorithm can fail, we have found it to be of great practical value due to the following reasons:

- It takes only a few milliseconds to solve a problem of 100 nodes. Thus it is more efficient to repeatedly run this algorithm, with different random starting solutions, in the hope that eventually an answer may be obtained rather than use other large scale algorithms [2,4,7,8].

- The algorithm has solved problems having 30,000 nodes. Such a large problem takes less than one minute of DEC-20 CPU time. Our method is constrained by memory limitations and not by computation time. It would be possible to overcome the memory limitation by packing data or by making use of secondary storage devices.

The original title of this paper was "A polynomial algorithm which has positive probability of solving a Hamiltonian path problem".

It was prepared as part of the activities of the Management Science Research Group, Carnegie-Mellon University, under contract No. N00014-82-K-0329 NR 047-048 with the U.S. Office of Naval Research. Reproduction in whole or part is permitted for any purpose of the U.S. Government.

### 2. Exact statement of the problem

Let  $G = (V, A)$  be a directed graph with  $V$  as the set of nodes and  $A$  as the set of arcs.  $G$  has  $n$

nodes and they are numbered sequentially from 1 to  $n$ . The indegree of node 1 is zero. The outdegree of node  $n$  is also zero. The problem is to find the HP from node 1 to node  $n$  which goes through all the intermediate nodes exactly once.

### 2.1. The Hamiltonian cycle problem

The problem of finding a Hamiltonian cycle in a directed graph can be easily converted to a problem of finding a HP as shown in Figure 2.1. We split any node of the graph into two nodes so that one node has no incoming arc and the other has no outgoing arc. The Hamiltonian path between these two nodes would be a Hamiltonian cycle in the original graph. In Figure 2.1 HP between nodes 1 and 1a is equivalent to a HC. Thus we are solving the general Hamiltonian path problem rather than that for a specific type of graph.

### 2.2. The postman pickup problem

This is a problem which can be translated into a Hamiltonian path problem. A postman has to pick up one letter from each of  $n$  locations and deliver them to the main post office. Starting from his home, how can he do this while covering the minimum possible distance?

Let node 1 represent his home and node  $n$  represent the post office. Nodes 2 through  $n-1$  are the locations from which letters have to be picked up. The Hamiltonian path from node 1 to node  $n$  is a possible path on which the postman should travel in order to minimize his total distance traveled.

### 3. Preliminary concepts and definitions

We give some preliminary definitions with interpretations related to the postman pickup problem.

**Definition 1.** An *arborescence* is a directed tree having one node called the *root node* which can be reached by a unique directed path from any node.

For the postman pickup problem, the root node is the post office.

**Definition 2.** Let  $G = (V, A)$  be a directed graph. Then arborescence  $T = (V, A_T)$  is a *spanning arborescence* of  $G$  if  $A_T$  is a subset of  $A$ .

**Definition 3.** Let  $i$  and  $j$  be two nodes of a spanning arborescence of  $G$  and  $(i, j) \in A_T$ .

1. If there exists a directed path in  $T$  from  $i$  to  $j$  then  $(i, j)$  is said to be an *up arc*.
2. If there exists a directed path in  $T$  from  $j$  to  $i$  then  $(i, j)$  is said to be a *down arc*.
3. Otherwise  $(i, j)$  is called a *cross arc*.

**Definition 4.** If  $i, j \in V$  and  $(i, j) \in A_T$ , then the *predecessor node* of  $i$  is said to be  $j$  or symbolically  $p(i) = j$ .

One can interpret predecessor of  $i$  to be like the 'father' of  $i$  in the sense of a family tree.

**Definition 5.** A node  $i \in V$  is said to be a *junction node* of arborescence  $T = (V, A_T)$  if it is the end node of at least two arcs belonging to  $A_T$ .

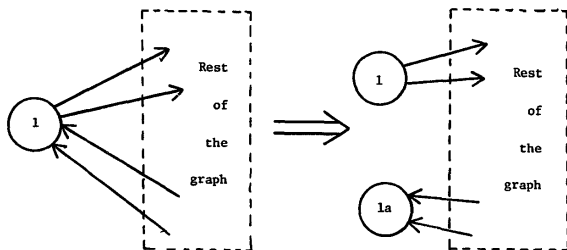


Fig. 2.1. Converting a Hamiltonian cycle problem to a Hamiltonian path problem.

**Definition 6.** A node  $i \in V$  is said to be a *beginning node* if it is not the end node of any arc  $\in A_T$ .

**Definition 7.** For each  $i \in V$  we define the *successor set*  $\Psi_i = \{h: p(h) = i\}$ . Thus  $\Psi_i$  is the set of nodes which are immediately below  $i$ , i.e. the set of nodes for which  $i$  is the predecessor node.

**Lemma 8.** If  $\|\Psi_i\| = 0$  then  $i$  is a *beginning node*.

**Lemma 9.** If  $\|\Psi_i\| \geq 2$  then  $i$  is a *junction node*.

**Definition 10.** The *successor function* of  $i$  represented by  $s(i)$ , for  $i \in V$  is defined inductively as follows:

$$s(i) = 1 + \sum_{h \in \Psi_i} s(h).$$

In other words,  $s(i) = 1 +$  number of nodes in  $T$  below  $i$ . For the postman pickup problem  $s(i)$  is the number of letters the postman will carry after leaving node  $i$ .

**Lemma 11.** If  $i$  is a *beginning node* then  $s(i) = 1$ .

**Definition 12.** The *ramification index*  $R$  of an arborescence  $T = (V, A_T)$  is defined as follows:

$$R \equiv R(T) = n * (n-1)/2 - \sum_{i \in V - \{n\}} s(i).$$

**Lemma 13.** The *maximum possible ramification index* of an arborescence is  $(n-1) * (n-2)/2$ .

**Proof.** The arborescence for which the ramification index is maximum is the one for which the only junction node is the root node and each of the  $n-1$  arcs in  $A_T$  has the root node as its end node. For this case the values of the successor function at each node  $i$  for  $i = 1, \dots, n-1$  is 1 and we get from Definition 12:

$$\begin{aligned} R &= n * (n-1)/2 - (n-1) \\ &= (n-1) * (n-2)/2 \end{aligned}$$

**Definition 14.** Let  $J$  be the set of junction nodes of arborescence  $T$ . The *ramification index* of a junction node  $j \in J$  is defined as:

$$R_j = \sum_{i_1, i_2 \in \Psi_j, i_1 < i_2} s(i_1) * s(i_2).$$

The significance of Definition 14 is due to the following theorem.

**Theorem 15.** The *ramification index* of an arborescence  $T$  can be computed from the following equation:

$$R \equiv R(T) = \sum_{j \in J} R_j.$$

**Proof.** Assume  $T = (V, A_T)$  has  $n$  nodes. We will first prove the proposition for the case in which  $T$  has only one junction node  $j$ . The nodes above  $j$  do not contribute to the ramification index because, as will be seen in Theorem 16 ( $4 \Rightarrow 5$ ), a tree with no junction nodes has zero ramification index. Let  $s_1$  to  $s_m$  be the successor functions of the  $m$  arcs coming into  $j$ . Note that  $j = n$  implies  $s_1 + s_2 + \dots + s_m = n-1$ . According to Definition 12,

$$R \equiv R(T) = n * (n-1)/2 - \sum_{i \in V - \{n\}} s(i)$$

or

$$\begin{aligned} R_j &= R = \sum_{i=1}^{n-1} i - \sum_{j=1}^m \sum_{i=1}^s i \\ &= \left[ \sum_{i=1}^{s_1} i - \sum_{i=1}^{s_1} i \right] + \left[ \sum_{i=s_1+1}^{s_1+s_2} i - \sum_{i=1}^{s_2} i \right] + \dots \\ &\quad + \left[ \sum_{i=s_1+\dots+s_{m-1}+1}^{s_1+\dots+s_m} i - \sum_{i=1}^{s_m} i \right] \\ &= s_1 * s_2 + (s_1 + s_2) * s_3 + \dots \\ &\quad + (s_1 + s_2 + \dots + s_{m-1}) * s_m. \end{aligned}$$

The last expression is equivalent to the expression in Definition 14.

We have proved the proposition when there is just one junction node in  $T$ . By induction, assume that the proposition is true for up to  $k-1$  junction nodes in  $T$ . We will now show that the proposition is also true for  $k$  junction nodes in  $T$ .

Let  $J$  be the set of  $k$  junction nodes of  $T$  with  $j$  designating the top most junction node. We again assume that  $j = n$ . The proof is stated for the case in which  $(i_1, j)$  and  $(i_2, j)$  are the only two arcs coming into  $j$ . Let successor functions of  $i_1$  and  $i_2$  be  $s_1$  and  $s_2$  respectively. Let  $V_1$  denote all the nodes below and including  $i_1$  and  $V_2$  denote all the nodes below and including  $i_2$ . Furthermore, let  $J_1$  and  $J_2$  denote the sets of junction nodes below  $i_1$

and  $i_2$  respectively. Then by Definition 12

$$R(T) = (s_1 + s_2 + 1)(s_1 + s_2)/2 \\ - \sum_{i \in V_1} s(i) - \sum_{i \in V_2} s(i).$$

Since  $|J_1| \leq k-1$  and  $|J_2| \leq k-1$ , we can write

$$R(T) = (s_1 + s_2)(s_1 + s_2 + 1)/2 \\ - \left[ s_1(s_1 + 1)/2 - \sum_{j \in J_1} R_j \right] \\ - \left[ s_2(s_2 + 1)/2 - \sum_{j \in J_2} R_j \right] \\ = s_1 * s_2 + \sum_{j \in J_1} R_j + \sum_{j \in J_2} R_j \\ = \sum_{j \in J} R_j.$$

This completes the proof.

If there were more than two arcs entering  $j$ , the proof would simply require more summation terms.

**Theorem 16.** Let  $T = (V, A_T)$  be a spanning arborescence of a directed graph  $G = (V, A)$ . The following statements are equivalent:

1.  $T$  is a Hamiltonian path starting from node 1 and ending at node  $n$ .
2. There is a directed path in  $T$  from node 1 to any other node  $x \in V$ .
3.  $T$  has node 1 as its only beginning node.
4.  $T$  has no junction nodes.
5.  $T$  has zero ramification index.

**Proof.** The proofs for the implications  $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4$  are obvious.

$4 \Rightarrow 5$ : The assertion follows directly from Theorem 15. Because  $J = \emptyset$  there are no terms in the first summation and thus the ramification index must be zero.

$5 \Rightarrow 1$ : Because the ramification index of  $T$  is 0, from Definition 12, we have

$$\sum_{i \in V - \{n\}} s(i) = n * (n - 1)/2.$$

Since all  $s(i)$  are integer, the only way in which this is possible is if there is an ordering of the nodes with  $s(i_1) = 1, s(i_2) = 2, \dots, s(i_{n-1}) = n - 1$ . Because the number of successors increases sequentially, it follows that  $T$  must be a Hamiltonian path. Because the indegree of node 1 is 0,

$i_1 = 1$ . Also because the outdegree of  $n$  is 0 we can choose  $i_n = n$ .

#### 4. A polynomial algorithm which has positive probability of finding a Hamiltonian path (if there is one)

Let  $G = (V, A)$  be a directed graph with  $n$  nodes as described in Section 2. The algorithm to be described makes use of Theorem 16 which says that the ramification index of a Hamiltonian path is zero. We first find a spanning arborescence of  $G$  denoted by  $T$  by constructing a greedy starting solution. Call the exchange of one arc in the arborescence with one arc not in the arborescence as one pivot. By performing a sequence of pivots we try to reduce the ramification index of  $T$  and when (and if) the ramification index of  $T$  becomes 0, we have found a Hamiltonian path.

The algorithm can get stuck at a given arborescence  $T$  with  $R(T) > 0$  if it is unable to find a pivot which will result in a decrease in the ramification index. This may mean: (a) that the graph has no Hamiltonian path, or (b) that the algorithm made an unfortunate choice of pivots which led it to get stuck. The alternative to stopping in this case is to choose a different starting solution and go through the algorithm again. It can be easily shown that there exists a starting solution which will lead to a Hamiltonian path, if there is one, after a finite number of pivots.

For ease of exposition, we present the algorithm in three separate parts.

##### 4.1. Algorithm 1: Finding an initial spanning arborescence

We want to find a spanning arborescence  $T = (V, A_T)$  of a directed connected graph  $G = (V, A)$ . For this purpose we define  $M$  which we call the set of marked nodes. This algorithm begins with  $A_T = \emptyset$  and ends when set  $A_T$  has been completely defined.

**Step 0:** Set  $M = \{n\}$ ,  $A_T = \emptyset$ , and  $k = 0$ .

**Step 1:** Choose any arc  $(i, j) \in A$  such that  $i \notin M$  and  $j \in M$ .

**Step 2:** Set  $M = M \cup \{i\}$ ,  $A_T = A_T \cup \{(i, j)\}$  and  $k = k + 1$ .

**Step 3:** If  $k = n - 1$  then STOP, else go to Step 1.

The above is the greedy form of the algorithm

for finding a spanning arborescence because it chooses as many as possible of those arcs to enter the solution which have their end points already marked to belong to  $T$ . A random greedy starting arborescence can be found by letting  $p$  be any number satisfying  $0 < p < 1$  and altering Step 2 as follows:

**Step 2\*:** Let  $x$  be a random number between 0 and 1: if  $x < p$  go to step 1, otherwise set  $M = M \cup \{j\}$  and  $A_T = A_T \cup \{(i, j)\}$ .

#### 4.2. Algorithm 2: Calculating the change in ramification index

Given a spanning arborescence, it is possible to compute its ramification index either by using Definition 12 or else by using the equations of Theorem 15. However we now present a method of calculating the change in ramification index which is much faster than those methods.

Given an arborescence, a cross arc  $(i, j)$  is to be brought into the solution. Let there be  $m_1$  arcs on the path from node  $i$  to node  $n$  and  $m_2$  arcs on the path from node  $j$  to node  $n$ . Let the successor function of  $i$  be denoted by  $\delta$ . Noting that an increase in successor function means an equal decrease in the ramification index, consider what happens when the cross arc  $(i, j)$  is brought into the solution:

1. The successor function of each of the nodes on the path from node  $i$  to node  $n$  (excluding node  $n$ ) goes down by  $\delta$  so that the corresponding change in ramification index is

$$\Delta R_1 = m_1 * \delta. \quad (1)$$

2. The successor function of each of the nodes on the path from node  $j$  to node  $n$  (excluding node  $n$ ) goes up by  $\delta$  so that the corresponding change here is

$$\Delta R_2 = -m_2 * \delta. \quad (2)$$

3. Finally, the extra arc  $(i, j)$  is added to the arborescence giving us

$$\Delta R_3 = -\delta. \quad (3)$$

The sum of equations 1 to 3 is the total change in ramification index if arc  $(i, j)$  is brought into the solution. We have now proved the following theorem.

**Theorem 17.** *The change in ramification index of an arborescence due to the introduction of a cross arc*

*$(i, j)$  into the solution is given by*

$$\Delta R = (m_2 - m_1 - 1) * \delta$$

where

$m_1$  is the number of arcs on the path from node  $i$  to node  $n$ ,

$m_2$  is the number of arcs on the path from node  $j$  to node  $n$ , and

$\delta$  is the successor function of node  $i$  before arc  $(i, j)$  is brought into the solution.

Now Algorithm 2 can be stated very simply:

**Step 1:** Find a cross arc.

**Step 2:** Using Theorem 17 find the change in ramification index if that arc were to be brought into the solution.

#### 4.3. Algorithm 3: Finding a Hamiltonian path

**Step 0: Find initial solution.** Find a directed rooted spanning arborescence of  $G$  with  $n$  as the root node by using Algorithm 1.

**Step 1: Calculate the successor function and ramification index of starting arborescence**  $T = (V, A_T)$ . Using Definition 10 calculate  $s(i)$  for all  $i \in V$ . Then from Definition 12 or from Theorem 15 calculate  $R(T)$ . If  $R(T) = 0$ : STOP -  $T$  is a Hamiltonian path. Else go to next step.

**Step 2: Find the incoming arc.** For each cross arc  $(i, j) \in A - A_T$  calculate the ramification index if that arc is brought into  $A_T$  using Algorithm 2. Let  $R_{\text{new}}$  be the minimum of these ramification indexes and let  $(i_1, j_1)$  be the corresponding arc.

**Step 3: Check possible cases.** If

$R_{\text{new}} = 0$ : STOP - A Hamiltonian path has been found.

$R_{\text{new}} \geq R$ : STOP - The algorithm failed to find a Hamiltonian path on this trial. If another trial is desired go back to Step 0 and generate a new, different spanning arborescence.

$R_{\text{new}} < R$ : Go to next step.

**Step 4: Perform pivoting.** Bring the cross arc  $(i_1, j_1)$  having ramification index  $R_{\text{new}}$  into the solution by setting  $A_T = A_T - \{(j_1, k) \in A_T\} \cup \{(i_1, j_1)\}$ . Let  $R = R_{\text{new}}$ . Update the successor function. Go to Step 2.

**Theorem 18.** *Algorithm 3 is a polynomial algorithm.*

**Proof.** The maximum possible loop value of an arborescence is of the order of  $O(n^2)$  [Lemma 13].

We decrease the loop value by at least 1 in each pivot, and each pivot step is linear in the number of arcs plus the number of nodes, so that Algorithm 3 is also  $O(n^2)$ .

#### 4.4. Algorithm 4: Finding a maximal $R$ spanning arborescence

It is interesting to note that with minor modifications Algorithm 3 can be used to find a spanning arborescence with maximal ramification index. The modified algorithm is described now.

**Step 0: Find initial solution.** Find a spanning arborescence of  $G$  using Algorithm 1.

**Step 1: Calculate successor function and ramification index of the starting arborescence  $T = (V, A_T)$ .** Using Definition 10 calculate  $s(i)$  for all  $i \in V$ . Then from Definition 12 or from Theorem 15 calculate  $R(T)$ .

**Step 2: Find incoming arc.** For each cross arc  $(i, j) \in A - A_T$  calculate the ramification index if that arc is brought into  $A_T$  using Algorithm 2. Let  $R_{\text{new}}$  be the maximum of these ramification indexes and let  $(i_1, j_1)$  be the corresponding arc.

**Step 3: Check possible cases.** If

$R_{\text{new}} \leq R$ : STOP - A maximal  $R$  spanning arborescence has been found.

$R_{\text{new}} < R$ : Go to next step.

**Step 4: Perform pivoting.** Bring the cross arc  $(i_1, j_1)$  having ramification index  $R_{\text{new}}$  into the solution by setting  $A_T = A_T - \{(j_1, k) : (j_1, k) \in A_T\} \cup \{(i_1, j_1)\}$ . Let  $R = R_{\text{new}}$ . Update the successor function. Go to step 2.

There are applications in which spanning arborescences which have maximal ramification index can be useful. One such application is the design of computer terminal networks in which signal concentrators are located at junction nodes [6].

It is obvious that a random choice of incoming arcs can be made in Step 2 of each of the last two algorithms by letting  $R_{\text{new}}$  be any incoming arc which decreases the ramification index in Algorithm 3 (or increase the ramification index in Algorithm 4).

## 5. Computational experience

We ran Algorithm 3 on Directed Rectangular Lattice Graphs (DRLG) because they are easy to

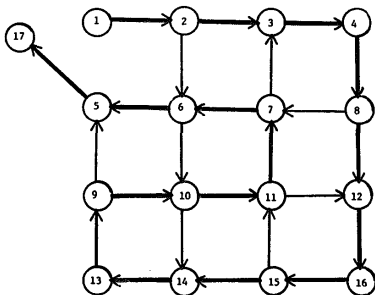


Fig. 5.1. Example of a 4x4 directed rectangular lattice graph. A Hamilton path is indicated by the darkened edges.

generate and it is possible to predict whether there is or is not a Hamilton path in a given DRLG. DRLG's are planar graphs which look like a grid as shown in Figure 5.1. They always have a Hamilton path when the number of nodes is even before node 1 is split. See [9] for details.

The computational experience is shown in Table 5.1.

We now make some observations about the computational results.

- The algorithm never failed on any DRLG which had a Hamiltonian path.

- The algorithm was coded in FORTRAN and all runs were performed on the DECSYSTEM-20 at Carnegie-Mellon University.

- The starting value of ramification index is about 98% of the maximum possible value of ramification index [Lemma 13]. A large value of  $R$  indicates a bushy tree which is a good start since it has less chance of getting stuck.

- According to Theorem 18, the upper bound on the number of pivots is the maximum possible ramification index. However, we can see from Tables 5.1 and 5.2 that the number of pivots required to obtain the solution is much less than the starting value of the ramification index.

- When a random starting solution is chosen, Algorithm 3 takes little more than twice the computing time compared to when a greedy starting solution is used. The number of pivots required is also double. Why this is to be expected is explained next.

Table 5.1.

Computational experience for Algorithm 3 with a greedy start

| Grid | # Nodes | # Arcs | Starting $R$ | # Pivots | CPU Time (sec) |
|------|---------|--------|--------------|----------|----------------|
| 10   | 100     | 180    | 4080         | 24       | 0.062          |
| 20   | 400     | 760    | 72 360       | 99       | 0.142          |
| 30   | 900     | 1740   | 378 840      | 224      | 0.292          |
| 40   | 1600    | 3120   | 1 217 520    | 399      | 0.631          |
| 50   | 2500    | 4900   | 3 002 400    | 624      | 1.138          |
| 60   | 3600    | 7080   | 6 267 480    | 899      | 1.868          |
| 70   | 4900    | 9660   | 11 666 760   | 1224     | 2.792          |
| 80   | 6400    | 12 640 | 19 974 240   | 1599     | 4.113          |
| 90   | 8100    | 16 020 | 32 083 920   | 2024     | 5.688          |
| 100  | 10 000  | 19 800 | 49 009 800   | 2499     | 8.118          |
| 110  | 12 100  | 23 980 | 71 885 880   | 3024     | 10.365         |
| 120  | 14 400  | 28 560 | 101 966 160  | 3599     | 14.232         |
| 130  | 16 900  | 33 540 | 140 624 640  | 4224     | 17.531         |
| 140  | 19 600  | 38 920 | 189 355 320  | 4899     | 21.587         |
| 150  | 22 500  | 44 700 | 249 772 200  | 5624     | 26.204         |
| 160  | 25 600  | 50 800 | 323 609 280  | 6399     | 39.976         |
| 170  | 28 900  | 57 460 | 412 720 560  | 7224     | 43.856         |
| 174  | 30 276  | 60 204 | 453 079 992  | 7568     | 38.434         |

- A greedy starting solution for a DRLG happens to be a good starting solution having a structure reasonably close to a Hamiltonian path. As can be seen from Table 5.1, it is even possible to predict the number of pivots required by a subsequent problem using the formula

Change in number of pivots

= One-fourth the change in number of nodes.

For a random start, there is no regular pattern.

- Number of CPU seconds required per pivot per node is a constant indicating empirically that each pivot requires a linear amount of work.

- The CPU times given here cannot be compared with other computational results. Reference [3] has computational experience for problems having up to only 60 nodes whereas [6] has no computational experience.

Table 5.2

Computational results for random starting solution

| # Nodes | # Arcs | Starting $R$ | # Pivots | CPU Time (sec) |
|---------|--------|--------------|----------|----------------|
| 100     | 180    | 4064         | 46       | 0.118          |
| 400     | 760    | 72 008       | 177      | 0.442          |
| 900     | 1740   | 337 488      | 481      | 1.773          |
| 1600    | 3120   | 1 213 976    | 852      | 7.474          |
| 2500    | 4900   | 2 994 828    | 1323     | 11.965         |
| 3600    | 7080   | 6 251 360    | 2151     | 22.966         |
| 4900    | 9660   | 11 654 952   | 2889     | 35.652         |
| 6400    | 12 640 | 19 949 896   | 3994     | 86.459         |

## 6. Conclusions

- We have devised a way of improving the performance of Algorithm 2 which is expected to decrease the CPU times noted in Tables 5.1 and 5.2.

- We will present a proof elsewhere that Algorithm 3 will never fail when applied to a DRLG with an even number of nodes (before any transformations are performed). A DRLG with odd number of nodes does not have any HP so that the algorithm will always fail.

- Algorithm 3 as stated is very inefficient for undirected graphs. We have developed a modification of Algorithm 3 which has proved to be very effective for finding a Hamiltonian path on an undirected graph. The results will be reported in a later paper.

## References

- [1] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North-Holland, 1978.
- [2] N. Christofides and S. Eilon, "Algorithms for large-scale travelling salesman problems", *Op. Res. Q.* **23** 511-518 (1972).
- [3] N. Christofides, *Graph Theory - An Algorithmic Approach*, Academic Press, London, 1975.
- [4] H. Crowder and M.W. Padberg, "Solving large scale symmetric travelling salesman problem to optimality", *Management Science* **26**, 495-509 (1980).

- [5] R.S. Garfinkel and G.L. Nemhauser, *Integer Programming*, Wiley, New York, 1972.
- [6] B. Gavish, "Topological design of centralized computer networks - Formulations and algorithms," *Networks* **12** 355-377 (1982).
- [7] T.H.C. Smith, "A LIFO implicit enumeration algorithm for the asymmetric travelling salesman problem, using a one-arborescence relaxation", Ph.D. Th., Carnegie-Mellon University, 1975.
- [8] T.H.C. Smith and G.L. Thompson "A LIFO implicit search algorithm for the symmetric travelling salesman problem using Held and Karp's one tree relaxation," *Annals of Discrete Mathematics* **1**, 495-506 (1977).
- [9] G.L. Thompson, "Hamiltonian tours and paths in rectangular lattice graphs," *Mathematics Magazine* **50**, 147-150 (1977).